Notices

First Edition (November 1994)

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Requests for technical information about IBM products should be made to your IBM authorized reseller or IBM marketing representative.

Copyright Notices

Disclaimers

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights or other legally protectable rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, programs, or services, except those expressly designated by IBM, are the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood NY 10594, U.S.A.

Trademarks

The following terms, denoted by an asterisk (*) in this publication, are trademarks of the IBM Corporation in the United States or other countries:

IBM
Micro Channel
OS/2
PC-XT

The following terms, denoted by a double asterisk (**) in this publication, are trademarks of other companies as follows. Other trademarks are trademarks of their respective companies.

Adaptec	Adaptec, Inc.
PCMCIA	Personal Computer Memory Card International Association
Pro AudioSpectrum	MediaVision, Inc.
Windows	Microsoft Corporation

About This Book

*Resource Manager for OS/2** contains a description of the OS/2 Resource Manager Architecture. Areas covered include architecture overview, programming interface definitions, programming structure definitions, and general development usage information. This document is written for device driver developers using OS/2.

The chapters include the following:

Resource Management Architecture gives an overview of the Resource Manager and describes its architecture.

RESERVE.SYS describes how the RESERVE.SYS device driver uses the Resource Manager for determining the resources available for a device driver.

Linking Resource Manager Services discusses the programming considerations of adding Resource Management services to your driver.

Resource Manager Services describes physical node management, resource management, node searches, node information, and logical node management. It also has a section that describes the Resource Manager return codes.

Resource Manager IOCtls defines the two IOCTls that are used for Resource Manager.

RMBASE.H contains the header file for Resource Manager.

Notices contain IBM and non-IBM trademarks used in this document.

Conventions Used in This Book

Unless otherwise stated, "OS/2" as used in this book refers to the Warp version of the IBM* Operating System/2.

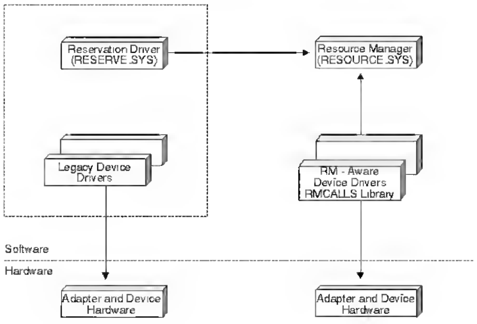
Resource Management Architecture

This chapter contains a description of the OS/2 Resource Management architecture.

Overview

The increased user demand for additional functions in a computer system has resulted in a broader range of more sophisticated peripheral and internal system devices. The continued expansion of computer devices has in turn, stressed the already limited pool of system resources, as well as raised the complexity of detecting and configuring the devices.

Therefore, OS/2 is introducing a centralized Resource Management architecture to facilitate the coexistence and cooperation of the increasing number of device drivers. The following figure depicts the main components that comprise the OS/2 Resource Management architecture.


Reservation Driver (RESERVE.SYS)
Resource Manager (RESOURCE.SYS)
Legacy Device Drivers
RM - Aware Device Drivers RMCALLS Library
Software
Hardware
Adapter and Device Hardware
Adapter and Device Hardware

With the introduction of this architecture, a functional line has been drawn separating previous device driver capabilities and those of the OS/2 device drivers written to use the Resource Manager services. Device drivers written to use the Resource Manager Services are referred to as *RM-aware* drivers. Existing OS/2 2.x device drivers are referred to as *legacy* drivers. Furthermore, the definition of legacy drivers includes DOS, Windows**, and OS/2 2.x drivers.

At the center of this architecture is the Resource Manager (RESOURCE.SYS), which provides a set of C-callable services to the other RM-aware components. As denoted by the dotted box surrounding the legacy drivers in the preceding figure, an additional RM-aware component (RESERVE.SYS) has been added to report the resources used by these drivers. This reservation driver is key to maintaining an accurate map of the system resources being managed by the Resource Manager. (See RESERVE.SYS for more details on the reservation driver.)

The Resource Manager manages much more than just the system hardware resources. It also assumes centralized responsibility for coordinating all aspects of both the logical and physical views of the hardware and supporting software in the system. The logical view is defined as the standard aliases assigned to devices for application reference (such as COM1, drive A, and LPT1). The physical view is defined as the actual details of the hardware topology such as port addresses and bus type.

In order to discuss the organization of the logical and physical views provided by the Resource Manager, several concepts and terms need to be reviewed. These concepts and terms are presented next.

Resources

The hardware resources managed by the Resource Manager are:

- I/O Port Ranges
- IRQ Levels
- DMA Channels
- Memory Regions
- Timer Channels

Driver writers are encouraged to identify their hardware settings during driver initialization directly rather than relying on the end-user to provide this information.

The Resource Manager services facilitate this process by identifying resources (I/O Ports, DMA Channels, and IRQ Levels) that are already claimed by drivers and have been initialized previously. A driver using the Resource Management calls can avoid inadvertently corrupting the state of adapters that were initialized by previously loaded drivers.

When resources are assigned, various levels of sharing are allowed. The type of sharing allowed depends on the underlying characteristics of the devices. Resource sharing options are discussed in Managing System Resources.

Drivers, Adapters, and Devices

The Resource Manager also manages drivers, adapters, and devices. Adapters and devices are associated with each other in a parent-child-sibling relationship. A driver is associated with each adapter or device node. They are defined as follows:

Drivers

A driver usually represents a software module responsible for management of one or more pieces of physical hardware. In addition, drivers include value-added subsystems that may not directly interact with physical hardware. To use other Resource Manager services, it is necessary to provide a driver handle that is obtained when declaring a driver.

Adapters

Adapters are defined as devices that convert from one bus protocol to another. For example, a SCSI adapter converts the host bus protocol (for example, ISA, EISA, or PCI), to SCSI bus protocols. The system bus (for example, ISA, EISA, or PCI) is represented in the Resource Manager as an adapter, converting from the CPU-internal bus protocols to a standard host-bus protocol.

In some cases, adapters may not directly correspond to physical packaging. An example of this could be a CD-ROM port on a multifunction sound board that is represented as a separate adapter. However, it is generally encouraged to try to "align" adapters with the physical packaging of the system to help end-users to identify their hardware easily.

Devices

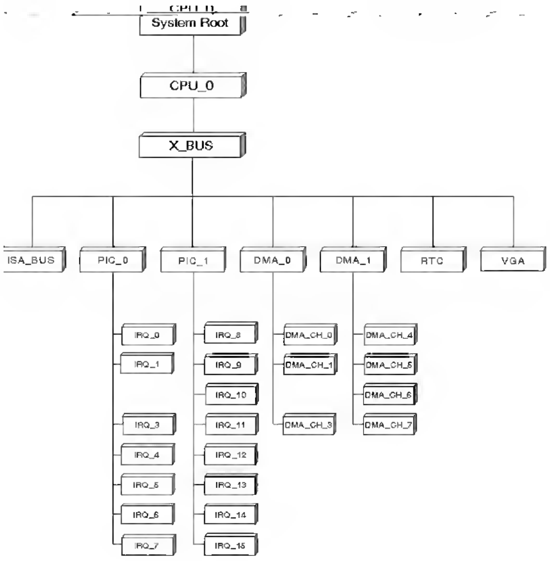
Devices are defined in the "traditional" meaning of end-user devices. This includes printers, disks, CD-ROMS, and so forth.

Physical View Management

The Resource Manager maintains a physical view of the system (hardware topology) that is utilized by all RM-aware components.

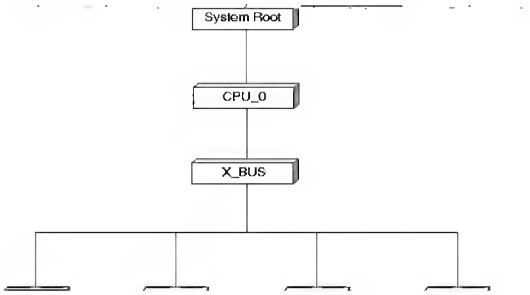
The physical view is shown in a tree structure. Each node within the tree structure represents a device or adapter. Associated with each adapter or device node are resource and driver structures.

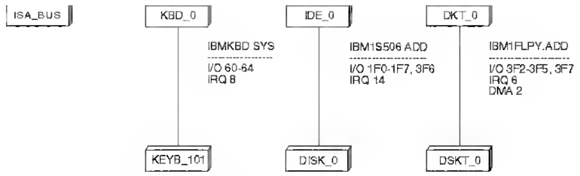
The Resource Manager automatically creates adapter and device nodes representing basic system resources, such as the CPU and system buses during its initialization. In addition, devices managed directly by the OS/2 kernel, such as the Interrupt Controller (PIC) and DMA Channels, are also automatically defined. In the following diagram, the X_Bus represents the system bus where many of the integrated components reside, such as the DMA control and the real-time clock. The following is a physical view of the Resource Manager.



As device drivers initialize, additional adapter and device nodes are created by the device drivers issuing Resource Manager service calls.

Each node has associated resources owned by the underlying device or adapter and the software driver responsible for managing the adapter or device. The following diagram illustrates the process.




ISA_BUS
KBD_0
IDE_0
DKT_0
IBMKBD SYS
I/O 60-64
IRQ 8
IBM1S506 ADD
I/O 1F0-1F7, 3F6
IRQ 14
IBM1FLPY.ADD
I/O 3F2-3F5, 3F7
IRQ 6
DMA 2
KEYB_101
DISK_0
DSKT_0

Device Adapter Keys

Each adapter device node contains text describing the adapter or device. To facilitate searching for a particular node, the beginning characters of the descriptive text are assumed to be keys. Keys are subject to the following rules:

- Any sibling nodes, such as nodes connected to the same parent, must have unique keys.
- Keys end at the first blank character or the 16th consecutive non-blank character.
- Keys are part of the descriptive text and must consist of printable ASCII characters.

Assign unique keys to adapter nodes based on their product name or function. For example:

CPU_0	CPU x486
IDE_0	Generic IDE Channel Adapter
AHA154X_0	AHA154X_0
AHA154X_1	Adaptec** 154X SCSI Adapters
PAS16	Pro AudioSpectrum** Audio Adapter

If multiple occurrences of the same type of adapter occurs, keys must be made unique by appending a _0, _1, ..., suffix to the key.

The Resource Manager services can automatically generate the _0, _1, ... suffix based on the adapter number supplied. Refer to RMCreateAdapter - Obtain an Adapter Handle for further information.

Assign the following the naming conventions for device nodes:

COMM_0	Serial Port
DISK_0	DASD Device
CDROM_0	CDROM Device
TAPE_0	Tape Device
PRINTER_0	Printer Device
SCANNER_0	Scanner Device

The Resource Manager services can automatically generate the _0, _1, ... suffix based on the device number supplied. This is done by including a "#" character at the end of the key. The RMCreateAdapter service and the RMCreateDevice service will replace the "#" character with the adapter or device number indicated. For example:

```
DSKT_#  - 1.44MB diskette drive
```

will be converted to:

```
DSKT_0  - 1.44 MB diskette drive
```

This process will occur provided that an adjunct structure containing the device number (ADJ_DEVICE_NUMBER) is passed on the RMCreateDevice service call. Adjunct data structures are discussed in Adjunct Data. Keys for SCSI devices should use the above device types followed by _(t,l) corresponding to the Target/Lun SCSI device. For example:

```
DISK_(0,0)
CDROM_(3,0)
```

RMParseSCSIInquiry - Build SCSI Device Description automatically generates appropriate keys for SCSI devices. An adjunct structure containing the Target/Lun device (ADJ_SCSI_TARGET_LUN) should be passed on the RMCreateDevice service call.

Adjunct Data

At times it is necessary to pass data to the Resource Manager services that can be relevant in one case but not in another call to the same service. In these cases, this optional data is passed in a linked list of structures called an Adjunct List.

The contents of the structures are determined by the adjunct types that are defined in RMBASE.H. A few of the more common adjunct types are listed as follows:

ADJ_DEVICE_NUMBER
Contains the zero-based unit number for the device being created. The Resource Manager will use this number to enumerate the key for the device.

ADJ_ADAPTER_NUMBER
Contains the zero-based adapter number for the adapter being created. The Resource Manager will use this number to enumerate the key for the adapter.

ADJ_SCSI_TARGET_LUN
Contains the SCSI Target/Lun for the device being created. The Resource Manager will generate a SCSI type enumeration suffix as described in Device Adapter Keys.

ADJ_ADD_UNIT
Contains the ADD/DM ADD Handle and Unit Handle assigned to the device being created.

Managing System Resources

When a device driver allocates resources, it specifies the level of *sharing* that the driver is willing to accept. The level of sharing selected depends on the characteristics of the hardware, such as its ability to coordinate usage of IRQ or DMA channels with other adapters, as well as the importance of other functions that may need the same IRQ level. Resources are allocated on a first-come first-serve basis.

If a driver requests a resource that has been previously allocated to another driver, then the sharing mode selected by the earlier driver is used to determine if the new driver's request will be granted.

The following resource sharing options are available:

EXCLUSIVE

The resource (such as the I/O Port and DMA Channel) is committed to the owner until it is explicitly released. Any other requests for this resource will be denied.

SHARED

The resource will be granted to any requester that also requests the resource as shared. This implies that the users of the resource can use it at any time without interfering with each other. An example of a shared resource would be a shared interrupt on a Micro Channel* or EISA bus machine.

MULTIPLEXED

The resource will be granted to any requester that also requests the resource as multiplexed. Multiplexed (unlike shared) implies that only one owner can actively be using the resource and that there is explicit notification between owners to control which requester is using the resource. In general, this sharing protocol is private to the multiplexed resource.

GRANTYIELD

The resource will be granted to any requester that also requests the resource as GRANTYIELD. GRANTYIELD implies that the owner of the resource is willing to participate in a sharing of a resource that is arbitrated by a Resource Manager.

Note: GRANTYIELD protocol is not currently in version 1.1 of RESOURCE.SYS.

Logical View Management

The Resource Manager maintains a logical view of the system device nodes that is utilized by applications, end-users, and device managers (or drivers performing similar functions). This section discusses the key aspects of the logical view.

OS/2 uses aliases to identify various physical devices. Typically, these aliases are short descriptive names that are familiar to most users, such as DISK_0, drive A or C, COM1, and LPT1.

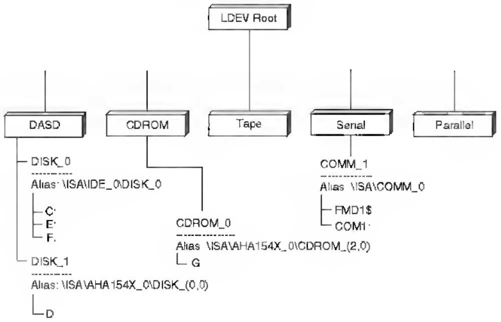
The use of aliases simplifies the operating system's (and end-user's) view of the system in the following ways:

- The implementation of a physical device (for example, a disk), depends on a combination of the system bus, a SCSI or IDE adapter, and the disk device. It is convenient to the end user to abstract this to a simple name such as DISK_0, DISK_1, and so forth, rather than using the full path name to the device.

- The operating system may subdivide (or group) devices so the physical device boundaries are no longer relevant. For example, partitioning subdivides a physical disk into a series of separate drives, while a RAID array may group several physical devices into a single drive.

- The operating system can have alternate views of the same physical device. For example, a physical serial port can be used as a FAX port (FMD1$) or used as an ASYNC port (COM1), depending on the application using the port.

To provide a mapping from the logical (end-user) view of the system to the actual hardware topology, the Resource Manager maintains a separate node structure called a logical tree as shown in the following diagram:



The logical tree contains two types of nodes, logical devices (LDEVs) and system names (SYSNAMEs). Logical device nodes represent aliases of nodes on the physical device tree. The system name nodes represent named divisions of the parent LDEV.

Depending on the nature of the device, these names can be alternate definitions of the LDEV that are mutually exclusive (such as FMD1$ and COM1:) or they can be shared (such as drive letters C, E, or F).

Logical devices and system names are associated with each other in a parent-child-sibling relationship. A driver is associated with each LDEV node.

The Resource Manager automatically creates logical device nodes representing basic system device types such as DASD, CD-ROM, tape, serial, and parallel, during its initialization. Device managers and certain device drivers (such as COM.SYS) create additional logical devices and system name nodes as they declare new devices to the OS/2 kernel.

RESERVE.SYS

RESERVE.SYS is used in conjunction with the Resource Manager in two separate scenarios:

- If you are using a device driver that is not Resource Manager-aware and you know the resources that the device uses, use RESERVE.SYS to reserve those resources so the Resource Manager-aware drivers will not have access to those resources.

- If you have a piece of hardware that does not tolerate the examination of its resources, reserve the resources so the Resource Manager-aware drivers do not examine the hardware.

To use RESERVE.SYS, place the following statement as the first line in CONFIG.SYS:

```
basedev=reserve.sys  <arguments>
```

Arguments:

```
    v
        /IO:
        /P:         /DW:                 /EXC
                                         /MUL
        /MEM:                            /SHA
        /DMA:
        /IRQ:
```

Switch	Format	Example	Description
/IO:	/IO:x,x	/IO:340,4	Reserve IO ports. The first number is the base port in HEX, and is followed by the

			length (number of ports) in decimal format.
/P:	same as /IO:		
/MEM:	/MEM:x,x	/MEM:CA00,1000	Reserve Memory. The first number is the base memory address (HEX), with the assumption that the address is XXXX:0, and is followed by the length (number of address) in decimal format.
/DMA:	/DMA:x	/DMA:2	Reserve DMA Channel. The number is in decimal format.
/IRQ:	/IRQ:x	/IRQ:13	Reserve IRQ. The number is in decimal format.
/EXC	/EXC	/EXC	Exclusive resource attribute.
/MUL	/MUL	/MUL	Multiplexed resource attribute.
/SHA	/SHA	/SHA	Shared resource attribute.
/DW:	/DW:x	/DW:10	Decode width of IO address. Valid numbers

are 10 and 16.
Only valid with
/IO: switch.

More than one resource attribute per-resource entry is an error and is not allowed. If no attributes or decode width is set, the default is EXCLUSIVE and 16. For example, to reserve IRQ 13 EXCLUSIVE, DMA 0 SHARED, MEMORY CA00:0 for 1000 bytes shared, IO ports 340 for 10 ports EXCLUSIVE and decode width 16, and IO ports 300 for 64 ports, with a decode width of 10:

```
BASEDEV=RESERVE.SYS /IRQ:13 /DMA:0 /SHA /MEM:CA00,1000 /SHA /IO:340,10 /IO:
```

Linking Resource Manager Services

This section discusses the programming considerations of adding Resource Management services to your driver.

The OS/2 Resource Manager consists of two components: RESOURCE.SYS and RMCALLS.LIB.

RESOURCE.SYS

RESOURCE.SYS is a base device driver. In OS/2 Warp and subsequent versions, this driver is provided as part of the product and is loaded automatically without an explicit CONFIG.SYS BASEDEV= statement.

RMCALLS.LIB

This library is linked with your device driver and provides the interface code to communicate with the RESOURCE.SYS driver. In addition, the RMCALLS library will provide rudimentary support for the following subset of Resource Manager services.

RMCreateDriver	RMDestroyDriver
RMCreateAdapter	RMDestroyAdapter
RMCreateDevice	RMDestroyDevice
RMCreateLDev	RMDestroyLDev
RMCreateSysName	RMDestroySysName
RMAllocResource	RMDeallocResource

In general, this support consists of returning a handle value of -1L and a return code of RMRC_SUCCESS as applicable. This support is enabled when your driver is run on a system that does not have RESOURCE.SYS installed, such as an earlier version of OS/2.

RMCALLS Library Data

The RMCALLS library references the following four variables:

```
     PFN Device_Help;
     ULONG RMFlags   = NULL;
     PFNRM RM_Help0  = NULL;
     PFNRM RM_Help3  = NULL;
```

The PFN Device_Help variable must be initialized by your driver prior to calling any Resource Manager services. It is expected to contain the Device Help entry point provided in the OS/2 Init Request Packet your driver receives.

Prior to calling any Resource Manager services, the remaining data variables must be initialized to zero. Specify C-initializers when declaring the variables.

These variables must be allocated by your driver. If you plan to use Resource Manager services after your driver has completed initialization, you must ensure that these variables are not discarded.

Note: If you do not declare these variables, a linker error message will occur, indicating that they are missing.

RMCALLS Library Code

The code portion of the RMCALLS library is included in a segment named RMCode. There are three alternatives in handling the code in this segment:

- Combine RMCode with your driver's default code segment if your driver does not intend to use Resource Manager services after initialization. Because the library code is linked after OBJ text, it would be discarded as your driver discards its initialization code.

- Combine RMCode with your driver's swappable code segment. If your driver intends to use Resource Manager services after its initialization and has a swappable code segment, then combine RMCode with this segment.

- Place RMCode in its own swappable segment. If your driver does not have a swappable code segment, then the RMCode will reside in its own swappable segment by default.

Resource Manager Services

The following Resource Manager services are grouped functionally and are provided by a library. The information in this library can be found in Linking Resource Manager Services.

Physical Node Management

The following services create driver, adapter, and device nodes:

- RMCreateDriver (see RMCreateDriver - Obtain a Driver Handle)
- RMCreateAdapter (see RMCreateAdapter - Obtain an Adapter Handle)
- RMCreateDevice (see RMCreateDevice - Obtain a Device Handle)

The following services delete driver, adapter, and device nodes:

- RMDestroyDriver (see RMDestroyDriver - Destroy a Driver Handle)
- RMDestroyAdapter (see RMDestroyAdapterHandle - Destroy an Adapter Handle)
- RMDestroyDevice (see RMDestroyDevice - Destroy a Device Handle)

Resource Management

The following services allocate or deallocate resources:

- RMAllocResource (see RMAllocResource - Obtain a Resource Handle)
- RMDeallocResource (see RMDeAllocResource - Destroy a Resource Handle)

The following service edits existing device or adapter resource sets by adding or removing resources:

- RMModifyResources (see RMModifyResources - Modify Adapter or Device Resource Sets)

Node Searches

The following service searches for nodes matching a particular key:

- o RMKeyToHandleList (see RMKeyToHandleList - Search for the Specified Adapter/Device/LDev Key)

The following service searches for nodes using a particular resource:

- o RMResToHandleList (see RMResToHandleList - Return List of Adapter/Device Handles That Own a Resource)

The following service searches for nodes containing matching adjunct data:

- o RMAdjToHandleList (see RMAdjToHandleList - Update an Adjunct Data Structure)

The following service returns the LDEV associated with a physical device node:

- o RMHDevtoHLDev (see RMHDevToHLDev - Return Physical Device Associated with Logical Device)

Node Information

The following service provides the type of node with which the handle is associated:

- RMHandleToType (see RMHandleToType - Return the Type of Resource Manager Handle)

The following service returns the handle of the node's parent:

- RMHandleToParent (see RMHandleToParent - Return a Parent Handle)

The following service returns the contents of the Resource Manager node indicated by the handle:

- RMGetNodeInfo (see RMGetNodeInfo - Return Resource Manager Node Information)

Logical Node Management

The following services create or destroy a logical device node:

- RMCreateLDev (see RMCreateLDev - Obtain a Logical Device Handle Adapter)
- RMDestroyLDev (see RMDestroyLDev - Destroy a Logical Device Handle)

The following services create or destroy a system name node:

- RMCreateSysName (see RMCreateSysName - Obtain a System Name Handle)
- RMDestroySysName (see RMDestroySysName - Destroy a System Name Handle)

The following service creates a *pseudo device* representing a group of physical devices:

- RMCreateLinkDevice (see RMCreateLDev - Obtain a Logical Device Handle Adapter)

Return Codes

This section describes Resource Manager return codes. All Resource Manager services return a 16-bit return code.

RMRC_SUCCESS

The Resource Manager service is successfully completed.

In cases where the Resource Management driver (RESOURCE.SYS) is not installed, some of the basic Resource Management services will return RMRC_SUCCESS, but do not perform an operation.

The purpose of the use of this return code, however, is to allow the use of the same device driver across various OS/2 versions without the driver having to check specific return codes indicating whether the Resource Manager is available.

RMRC_NOTINITIALIZED

A Resource Management library was not properly initialized. A device driver must call RMCreateDriver prior to issuing any other Resource Management service.

RMRC_BAD_DRIVERHANDLE

RMRC_BAD_ADAPTERHANDLE

RMRC_BAD_DEVICEHANDLE

RMRC_BAD_RESOURCEHANDLE

RMRC_BAD_LDEVHANDLE

RMRC_BAD_SYSNAMEHANDLE

The expected Resource Manager handles were not provided because the handle was not a valid Resource Manager handle or the handle did not point to the type of object the service required.

The individual return codes indicate the type of handle that was expected.

RMRC_BAD_DEVHELP

The Resource Management library requires the C-variable (Device_Help) to be initialized to the Device Help entry point prior to issuing the first Resource Management service call.

RMRC_NULL_POINTER

A Resource Manager service received a NULL value for a pointer that was expected to contain a valid

16:16 address.

RMRC_NULL_STRINGS
A descriptive text pointer in a DRIVERSTRUCT, ADAPTERSTRUCT, or DEVICESTRUCT datatype was found to be NULL rather than pointing to the expected ASCIIZ text data.

RMRC_BAD_VERSION
The Resource Manager level indicated on the RMCreateDriver service is not supported by the Resource Management driver currently installed because it is a a downlevel version of the Resource Management driver (RESOURCE.SYS), or the MajorVer/MinorVer fields of DRIVERSTRUCT were not properly initialized.

Refer to RMCreateDriver - Obtain a Driver Handle for further information.

RMRC_RES_ALREADY_CLAIMED
The requested resource is allocated exclusively to another driver, or the requested sharing mode conflicts with the sharing mode of other owners of the resource.

RMRC_INVALID_PARM_VALUE
A nonhandle or nonpointer variable contains an invalid or out-of-range value because:

- An invalid decode width was specified when allocating an I/O Port range.
- A handle search is being performed with cMaxHandles set to 0.

RMRC_OUT_OF_MEMORY
The Resource Manager is out of memory.

RMRC_BUFFER_TOO_SMALL
The buffer provided to receive information from a Resource Manager service was too small.

RMRC_IRQ_ENTRY_ILLEGAL
A Resource Manager service was issued at interrupt time. The Resource Manager service request can be issued only at task time or INIT time.

RMRC_NOT_IMPLEMENTED
The Resource Manager service requested is not implemented in the version of the Resource Manager you are currently using.

RMRC_NOT_INSTALLED
The Resource Management driver (RESOURCE.SYS) is required to service this request but is not installed.

RMADDToHDevice - Map an ADD/DM Handle to a Resource Manager Device Handle

This service converts an ADD/DM ADD Handle and ADD/DM UnitHandle to a corresponding Resource Manager device handle. This is done by searching the Resource Manager physical device tree for a node that contains an ADJ_ADD_UNIT adjunct, which matches the supplied ADD Handle and ADD UnitHandle.

Calling Sequence

```
rc = RMADDToHDevice( (PHDEVICE)  &hDevice,
                     (USHORT)    ADDHandle,
                     (USHORT)    UnitHandle );
```

Calling Parameters

```
&hDevice (PHDEVICE)
     Pointer to variable to receive device handle of device with a
     matching ADDHandle and UnitHandle.

ADDHandle (USHORT)
     Handle of the ADD driver assigned by DevHelp_RegisterDeviceClass.

UnitHandle (USHORT)
     UnitHandle the ADD driver assigned to this device.
```

Returns

On systems where the Resource Manager driver is not installed, the library interface code will return RMRC_NOT_INSTALLED and will set hDevice to -1L.

Remarks

Use RMAdjToHandleList rather than this service to maintain compatibility with future Resource Manager releases. This service will be removed in a future Resource Manager release.

Refer to the *Storage Device Driver Reference* for additional information on ADD/DM architecture.

RMAdjToHandleList - Update an Adjunct Data Structure

This service searches for Resource Manager nodes that match the adjunct structure specified. A list of node handles and adjunct indices are in the HandleList structure provided by the caller.

Calling Sequence

```
rc = RMAdjToHandleList( (PADJUNCT)       &AdjunctData,
                        (HADAPTER)       hStartNode
                        (PAJDHANDLELIST) &AdjHandleList );
```

Calling Parameters

&AdjunctData (PADJUNCT)

Pointer to an adjunct structure to search for.

The ADJUNCT type contains a union of data structures of varying lengths. When searching for a particular adjunct, set the length field of the AdjunctData structure that was passed, to the exact length of the adjunct structure being located. For example:

```
/* Correct */

 AdjData.AdjLength = ADJ_HEADER_SIZE + sizeof(ADD_UN

/* Incorrect */

 AdjData.AdjLength = sizeof(ADJUNCT);
```

hStartNode (HADAPTER)

Handle of resource manager node at which to start the search.

&AdjHandleList (PADJHANDLELIST)

Pointer to the following structure:

```
ADJHANDLELIST struct
              {
                USHORT        cMaxHandles;
                USHORT        cHandles;
                ADJINFO       Adj[1];
              };
```

```
ADJINFO struct
        {
          HADAPTER    hAdapter;
          USHORT      AdjIndex;
        };
```

cMaxHandles (USHORT)

Number of handles that can be accepted in the handles array.

This field must be set by the caller.

The default ADJHANDLELIST type provides room for one handle. In this case, cMaxHandles must be set to 1. Refer to the example in RMKeyToHandleList - Search for the Specified Adapter/Device/LDev Key, if more than one Adjunct handle is expected.

cHandles (USHORT)

Actual handle count found. This field must be initially set to zero by the caller.

The number of handles reported by this field may exceed cMaxHandles. This indicates that the HandleList structure supplied was too small. The count will be set to the number of handles actually found.

Adj[ ] (ADJINFO)

Array of ADJINFO structures. The ADJINFO structure contains the following fields:

hAdapter (HADAPTER)

Handle of the owner of the adjunct data that matches the adjunct specified.

AdjIndex (USHORT)

Index of the adjunct structure found.

Returns

On systems where the resource manager driver is not installed, the library interface code will return RMRC_NOT_INSTALLED.

RMAllocResource - Obtain a Resource Handle

This service allows a driver to register usage of a hardware resource.

Calling Sequence

```
rc = RMAllocResource( (HDRIVER)          hDriver,
                      (PHRESOURCE)       &hResource,
                      (PRESOURCESTRUCT) &ResourceStruct );
```

Calling Parameters

hDriver (HDRIVER)
Driver handle of the device driver requesting the resource.

&hResource (PHRESOURCE)
Pointer to the variable to receive the returned resource handle.

&ResourceStruct
Pointer to the following structure:

```
RESOURCESTRUCT    struct
                  {
                    ULONG ResourceType;
                    union
                    {
                      IORESOURCE   IOResource;
                      IRQRESOURCE  IRQResource;
                      MEMRESOURCE  MEMResource;
                      DMARESOURCE  DMAResource;
                      TMRRESOURCE  TMRResource;
                    };
                    ULONG  Reserved;
                  };
```

ResourceType (ULONG)
Type of resource being requested.

The resource structure contained in the union must match the type of resource indicated by this field.

Resource Requested	Resource Type	Resource Struct
I/O Port Range	RS_TYPE_I/O	IORESOURCE
IRQ Level	RS_TYPE_IRQ	IRQRESOURCE
BIOS Memory Region	RS_TYPE_MEM	MEMRESOURCE
DMA Channel	RS_TYPE_DMA	DMARESOURCE

Timer Channel	RS_TYPE_TIMER	TMRRESOURCE

IOResource (IORESOURCE)

For ResourceType = RS_TYPE_IO, the union contains the following structure:

```
IORESOURCE  struct
            {
              USHORT BaseIOPort;
              USHORT NumIOPorts;
              USHORT IOFlags;
              USHORT IOAddressLines;
            };
```

BaseIOPort

Specifies the start of the I/O Port range requested. If a search is being performed, then this field is considered the starting point for the search.

NumIOPorts

Specifies number of contiguous I/O Ports requested.

IOFlags

One of the following sharing mode flags must be specified:

RS_IO_EXCLUSIVE
RS_IO_MULTIPLEXED
RS_IO_SHARED
RS_IO_GRANT_YIELD

See Managing System Resources for a description of resource sharing modes.

If a device driver and supported adapter is capable of selecting a new I/O range and is participating in Grant-Yield protocols, then the following flag indicates the driver will accept reconfiguration requests for this resource:

RS_IO_RECONFIGURE

If a search is to be performed, the following flag must be specified:

RS_SEARCH

When a search is performed and is successful, the *BaseIOPort* field will be updated to indicate the start of the allocated I/O Port range.

The port range returned will be aligned on 2^n boundary >= NumIOPorts.

IOAddressLines

Specifies the number of address lines the adapter can decode. This field must contain the value 10 or 16.

Most current-generation 16-bit adapters decode all 16 I/O address lines. PC-XT* and some early 16-bit adapters decode only 10 address lines.

Adapters that do not fully decode appear in multiple places in the I/O address space because they ignore the high-order I/O address bits.

Note: Port allocations outside the ISA compatibility range (100h - 3FFh) must indicate IOAddressLines = 16.

IRQResource (IRQRESOURCE)

For ResourceType = RS_TYPE_IRQ, the union contains the following structure:

```
IRQRESOURCE  struct
             {
               USHORT  IRQLevel;
               USHORT  PCIIrqPin;
               USHORT  IRQFlags;
               USHORT  Reserved;
               PFNRMINTHANDLER pfnIntHandler;
             };
```

IRQLevel

Specifies requested IRQ Level (0-15).

PCIIrqPin

For PCI devices, set to one of the following values based on the PCI Interrupt Pin assigned to the device.

PCI Interrupt Pin	PCIIrqPin Value

None	RS_PCI_INT_NONE
A	RS_PCI_INT_A
B	RS_PCI_INT_B
C	RS_PCI_INT_C
D	RS_PCI_INT_D

For non-PCI devices, this field must be set to zero. This entry is provided for information only. No conflict determination occurs with this entry.

IRQFlags

One of the following sharing-mode flags must be specified:

RS_IRQ_EXCLUSIVE
RS_IRQ_MULTIPLEXED
RS_IRQ_SHARED
RS_IRQ_GRANT_YIELD

See Managing System Resources for a description of resource sharing modes.

pfnIntHandler

This field is reserved for future use and must be set to zero.

MEMResource (MEMRESOURCE)

For ResourceType = RS_TYPE_MEM, the union contains the following structure:

```
MEMRESOURCE  struct
             {
               ULONG  MemBase;
               ULONG  MemSize;
               USHORT MemFlags;
               USHORT ReservedAlign;
             };
```

MemBase

Specifies the start of the BIOS memory region requested as physical address. If a search is being performed, then this is the starting point for the search.

MemSize

Specifies the size of the BIOS memory region requested in bytes.

MemFlags

One of the following sharing-mode flags must be specified:

RS_MEM_EXCLUSIVE
RS_MEM_MULTIPLEXED
RS_MEM_SHARED
RS_MEM_GRANT_YIELD

See Managing System Resources for a description of resource-sharing modes. If a device driver and adapter are capable of selecting a new memory range and are participating in Grant-Yield protocols, then the following flag indicates the driver will accept reconfiguration requests for this resource:

RS_MEM_RECONFIGURE

If a search is to be performed, the following flag must be specified:

RS_SEARCH

When a search is performed and is successful, the *MemBase* field will be updated to indicate the start of the allocated memory range.

DMAResource (DMARESOURCE)

For ResourceType = RS_TYPE_DMA, the union contains the following structure:

```
DMARESOURCE struct
            {
              USHORT DMAChannel;
              USHORT DMAFlags;
            };
```

DMAChannel

Specifies the number of the DMA Channel requested (0-7).

DMAFlags

One of the following sharing-mode flags must be specified:

RS_DMA_EXCLUSIVE
RS_DMA_MULTIPLEXED
RS_DMA_SHARED
RS_DMA_GRANT_YIELD

See Managing System Resources for a description of resource-sharing modes.

TMRResource (TMRRESOURCE)

For ResourceType = RS_TYPE_TIMER, the union contains the following structure:

```
TMRRESOURCE  struct
             {
               USHORT TMRChannel;
               USHORT TMRFlags;
             };
```

TMRChannel

Specifies the number of the timer channel requested (0-2).

TMRFlags

One of the following sharing mode flags must be specified:

RS_TMR_EXCLUSIVE
RS_TMR_MULTIPLEXED
RS_TMR_SHARED
RS_TMR_GRANT_YIELD

See Managing System Resources for a description on resource-sharing modes.

Returns

On systems where the Resource Manager driver is not installed, the library interface code will return RMRC_SUCCESS and a resource handle of -1L.

Remarks

Prior to making any attempt to access the resource, a driver must issue this call and successfully obtain ownership of a resource. This call includes non-destructive (read-type) accesses.

For I/O and BIOS memory resources, RMAllocResource searches to locate an available I/O or memory region that is supported. The resource to be allocated is described in a RESOURCESTRUCT, which contains a C-union of structures that are resource-type specific. This service returns a resource handle (HRESOURCE) that is used to identify usage of the resource by this driver.

Resources are initially assigned to the driver that is issuing the RMAllocResource request. By using hardware probing, the driver determines if the hardware it intends to support is installed.

When the driver determines that a supported adapter is present, it registers an adapter (RMCreateAdapter) and assigns resource handles by providing a list of resource handles in the RMCreateAdapter service request.

If an adapter is not found, resources that are not assigned to a device or adapter must be released. See RMDeAllocResource - Destroy a Resource Handle.

RMCreateAdapter - Obtain an Adapter Handle

This service allows a driver to register an adapter with the Resource Manager. An adapter handle (HADAPTER) is returned by this service to identify the adapter. Information about the adapter being registered is passed in an ADAPTERSTRUCT.

When an adapter is registered, a list of resource handles representing hardware resources used by this adapter may be optionally passed.

Calling Sequence

```
rc = RMCreateAdapter( (HDRIVER)        hDriver,
                      (PHADAPTER)      &hAdapter
                      (PADAPTERSTRUCT) &AdapterStruct,
                      (HDEVICE)        hParentDevice,
                      (PAHRESOURCE)    &ResourceList  );
```

Calling Parameters

hDriver (HDRIVER)
Driver handle of the device driver creating this adapter.

&hAdapter (PHADAPTER)
Pointer to the variable to receive the returned adapter handle.

&AdapterStruct
Pointer to the following structure:

```
ADAPTERSTRUCT struct
              {
                PSZ          AdaptDescriptName;
                USHORT       AdaptFlags;
                USHORT       BaseType;
                USHORT       SubType;
                USHORT       InterfaceType;
                USHORT       HostBusType;
                USHORT       HostBusWidth;
                PADJUNCT     pAdjunctList;
                ULONG        Reserved;
              };
```

AdaptDescriptName (PSZ)
Pointer to an ASCIIZ string containing the adapter key and a brief description of the adapter. For example:

```
"FLOPPY_#  Diskette Controller"
```

Note: Up to the first 16 non blank characters are used as a key to locate this adapter. The RMCreateAdapter service will substitute the "#" character with the adapter number supplied in the adjunct list.

See Device Adapter Keys for additional information concerning device adapter keys.

AdaptFlags (USHORT)

Adapter attribute flags. Unused flags must be set to zero by the caller.

The following is a valid flag value:

AS_16MB_ADDRESS_LIMIT

The adapter does not support data transfers to storage above 16MB (24-bit addressing limit).

The following three fields provide a category of the adapter being created: Refer to RMBASE.H for a complete list of adapter categories.

BaseType (USHORT)
Indicates the general functional category of the adapter:

AS_BASE_MSD	Mass storage including disk, tape, CD-ROM, and so forth.
AS_BASE_COMM	Communications, including serial and parallel ports.
AS_BASE_PERIPH	System board components, including DMA controllers, interrupt controller, and so forth.

Subtype (USHORT)
Indicates the interface supported by the adapter. For example:

SCSI Adapter	AS_BASE_MSD/AS_SUB_SCSI
IDE Adapter	AS_BASE_MSD/AS_SUB_IDE
Serial Port	AS_BASE_COMM/AS_SUB_SERIAL
Parallel Port	AS_BASE_COMM/AS_SUB_PARALLEL

InterfaceType (USHORT)
Provides specific interface information. For example:

Parallel Port	AS_BASE_COMM/AS_SUB_PARALLEL/AS_INTF_ECP

The following two fields describe the host bus that the adapter supports. The fields also describe how the adapter is connected to the host system rather than how the adapter is connected to the devices it supports:

HostBusType
Describes how the adapter is attached to the host system, such as ISA, EISA, Micro Channel, or PCI.

HostBusWidth

Indicates the maximum width of host bus data transfers the adapter supports. Refer to RMBASE.H for values for these fields.

pAdjunctList (PADJUNCT)

Pointer to a linked list of adjunct data structures. The following adjunct structures should be included in the list:

ADJ_ADAPTER_NUMBER   This adjunct contains the zero-based adapter number for the adapter being created.

Reserved (ULONG)

Reserved.

hParentDevice (HDEVICE)

Indicates the handle of the parent adapter or device for the adapter being created. This field can be set to NULL, in which case the adapter will be assigned to the default system bus. If the system contains multiple buses such as ISA or PCI, the *HostBusType* field will be used to choose the appropriate bus.

&ResourceList (PAHRESOURCE)

Pointer to a structure containing a count and a list of resource handles to be assigned to this adapter.

```
AHRESOURCE   struct
             {
               ULONG     NumResource;
               HRESOURCE hResource[1];
             }
```

Returns

On systems where the Resource Manager driver is not installed, the library interface code will return RMRC_SUCCESS and an adapter handle of -1L.

Remarks

As an alternative, resources may be added after the adapter is created by using the RMModifyAdapter service. In this case, the ResourceList pointer should be set to NULL.

RMCreateDevice - Obtain a Device Handle

This service allows a driver to register a device with the Resource Manager and to assign the device to an adapter. A device handle (HDEVICE) is returned by this service to identify the device.

When a device is registered, a list of resource handles (representing hardware resources used by this device) can be assigned optionally to the device.

Information about the device being registered is passed in a DEVICESTRUCT, which is described below.

Calling Sequence

```
rc = RMCreateDevice( (HDRIVER)       hDriver,
                     (PHDEVICE)      &hDevice,
                     (PDEVICESTRUCT) &DeviceStruct,
                     (HADAPTER)      hParentAdapter,
                     (PAHRESOURCE)   pahResource);
```

Calling Parameters

hDriver (HDRIVER)
Handle of the device driver creating this device.

&hDevice (PHDEVICE)
Pointer to a variable to receive the returned device handle.

&DeviceStruct
Pointer to the following structure:

```
DEVICESTRUCT   struct
               {
                 PSZ      DevDescriptName;
                 USHORT   DevFlags;
                 USHORT   DevType;
                 PADJUNCT pAdjunctList;
               };
```

DevDescriptName (PSZ)
Pointer to an ASCIIZ string containing the device key and a brief description of the device. For example:

```
"DSKT_#  1.44MB Diskette Drive"
"DISK_#   Fixed Disk"
```

Note: Up to the first 16 non-blank characters are used as a key to locate this device.

The RMCreateDevice service will substitute the "#" character with the device number or SCSI Target/Lun supplied in the adjunct list.

See Device Adapter Keys for additional information concerning device and adapter keys.

DevFlags (USHORT)
Device attribute flags. Unused flags must be set to zero by the caller. The following is a valid flag value:

DS_REMOVEABLE_MEDIA

The device has removable media.

DevType (USHORT)

Device type. Valid device types include:

DS_TYPE_DISK	All direct access devices.
DS_TYPE_TAPE	Sequential access devices.
DS_TYPE_PRINTER	Printer device.
DS_TYPE_PROCESSOR	Processor-type device.
DS_TYPE_WORM	Write-Once-Read-Many device.
DS_TYPE_CDROM	CD-ROM device.
DS_TYPE_SCANNER	Scanner device.
DS_TYPE_OPT_MEM	Optical disk.
DS_TYPE_CHANGER	Changer device.
DS_TYPE_COMM	Communication devices.
DS_TYPE_ATAPI	ATAPI protocol device.
DS_TYPE_SCSI_ATT	SCSI bus attachment (bridge controller).
DS_TYPE_SOCKET	PCMCIA socket.
DS_TYPE_SLOT	ISA/Micro Channel/EISA/PCI bus slot.
DS_TYPE_PLANAR_CHIPSET	DMA/IRQ/TIMER controllers.
DS_TYPE_IO	Input/Output.
DS_TYPE_AUDIO	Audio device.
DS_TYPE_UNKNOWN	Unknown device type.

pAdjunctList (PADJUNCT)

Pointer to a linked list of adjunct data structures. The following adjunct structures should be included in the list:

ADD Drivers	ADJ_ADD_UNIT This adjunct contains the ADD handle and ADD unit number the ADD driver will assign to the device.
Non-SCSI Devices	ADJ_DEVICE_NUMBER This adjunct contains the zero-based device number for non-SCSI devices. For example, the nth CD-ROM is supported by an adapter.
SCSI Devices	ADJ_SCSI_TARGET_LUN This adjunct contains the SCSI target ID and Logical Device Number (LUN) assigned to the device.

hParentAdapter (HADAPTER)

Handle of the parent adapter for the device being created.

&ResourceList (PAHRESOURCE)

Pointer to a structure containing a count and a list of resource handles to be assigned to this device.

```
AHRESOURCE  struct
            {
              ULONG      NumResource;
              HRESOURCE  hResource[1];
            }
```

Returns

On systems where the Resource Manager driver is not installed, the library interface code will return RMRC_SUCCESS and an adapter handle of -1L.

Remarks

Resources are usually assigned to the adapter (parent) that owns the device. In some cases, resources are used specifically by a particular device rather than shared between multiple devices supported by an adapter. In this case, resources should be assigned to the device, as appropriate.

RMCreateDriver - Obtain a Driver Handle

This service registers basic information about calling the device driver with the Resource Manager. A driver handle (HDRIVER) is returned by this service and is required by other Resource Manager services to identify the requestor.

The first call to this service causes the Resource Manager interface code (the part that is linked to your device driver) to initialize. Therefore, this function is the first Resource Manager call a driver usually makes.

Information about the driver registering with the Resource Manager is passed in a DRIVERSTRUCT that is described below.

Calling Sequence

```
rc = RMCreateDriver( (PDRIVERSTRUCT) &DriverStruct,
                     (PHDRIVER)      &hDriver         );
```

Calling Parameters

&hDriver

Pointer to a variable to receive the returned driver handle.

&DriverStruct (PDRIVERSTRUCT)

Pointer to the following structure:

```
DRIVERSTRUCT  struct
              {
                 PSZ       DrvrName;
                 PSZ       DrvrDescript;
                 PSZ       VendorName;
                 UCHAR     MajorVer;
                 UCHAR     MinorVer;
                 DATESTAMP Date;
                 USHORT    DrvrFlags;
                 USHORT    DrvrType;
                 USHORT    DrvrSubType;
                 PFNRMCB   DrvrCallback;
              };
```

DrvrName (PSZ)

Pointer to ASCIIZ name of device driver. For example:

```
"IBM1FLPY.ADD"
"C:\MMOS2\MVPRODD.SYS"
```

DrvrDescript (PSZ)

Pointer to a brief ASCIIZ description of device driver. For example:

```
"ISA/EISA Diskette ADD Driver"
```

VendorName (PSZ)

Pointer to ASCIIZ vendor name For example:

```
"IBM Corporation"
```

MajorVer (UCHAR)

MinorVer (UCHAR)

The Resource Management interface level is the version of the Resource Manager services your driver is using. Currently, these fields must be set to CMVERSION_MAJOR and CMVERSION_MINOR, which are defined in RMBASE.H.

Note: This is not the version or level of your driver. If this field does not contain a valid value, the RMCreateDriver request will fail.

Date (DATESTAMP)

Structure containing the year/month/day to identify the version of your driver.

```
DATESTAMP struct
          {
            USHORT Year;
            UCHAR  Month;
            UCHAR  Day;
          };
```

Year (USHORT) Year as a 16-bit integer.

Month (UCHAR) Month as an 8-bit integer. Allowable range (1-12).

Day (UCHAR) Day of month as an 8-bit integer. Allowable range (1-31).

For example:

```
October 11, 1994

pDS->Date.Year  = 1994;
pDS->Date.Month = 10;
pDS->Date.Day   = 11;
```

DrvrFlags (USHORT)

Driver attribute flags. Unused bits must be set to zero by the caller.

Valid flag values (See also DRF_* in RMBASE.H).

DRF_STATIC   Driver is loaded when the system is restored.

DrvrType (USHORT)

DrvrSubType (USHORT)

These fields identify the type of interface your driver supports. The following categories are defined:

DRT_PCMCIA   A driver that conforms to PCMCIA specifications.

DRS_SOCKETSERV   A driver that supports the PCMCIA Socket Service interfaces.

DRS_CARDSERV   A driver that supports the PCMCIA Card Service interfaces.

DRS_CLIENT   A client driver that solely supports PCMCIA options.

Note: If a driver supports both PCMCIA and non-PCMCIA options, then the driver category reflects the non-PCMCIA usage.

DRT_ADDDM   A driver that conforms to the Adapter Device Driver (ADD) or Device Manager (DM) interfaces.

DRS_DM   An ADD or DM driver that acts as a device manager interfacing directly with the OS/2 kernel. For example, OS2DASD.DMD and OS2CDROM.DMD.

DRS_FILTER   An ADD or DM driver that acts as a filter.

DRS_ADD   An ADD or DM driver that acts as an adapter device driver. For example, IBM1FLPY.ADD and IBM1S506.ADD.

DRS_DM_TRANSPORT   A device manager whose primary purpose is to convert another protocol to ADD or DM protocols. For example, OS2ASPI.DMD converts Adaptec ASPI protocols to ADD/DM protocols.

DRT_OS2 Character or block device drivers that interface directly with the OS/2 kernel.

DRS_CHAR A driver that creates a character device intended for end-user usage.

DRS_BLOCK A driver that creates block devices intended for end-user usage.

DRS_APP_HELPER A driver that provides private services intended for a particular application rather than the end user directly.

DRT_AUDIO A driver that conforms to the OS/2 Multimedia APIs.

DRT_SERVICE A driver that provides services to other drivers. For example, RESOURCE.SYS.

DrvrCallback (PFNRMCB)

Not implemented in version 1.1 of Resource Manager. Must be NULL.

Returns

On systems where the Resource Manager driver is not installed, the library interface code will return RMRC_SUCCESS and a driver handle of -1L.

RMCreateLDev - Obtain a Logical Device Handle Adapter

This service allows a driver to register a logical device with the Resource Manager. A logical device handle (HLDEV) is returned by this service to identify the logical device.

Information about the logical device being registered is passed in an LDEVSTRUCT.

Calling Sequence

```
rc = RMCreateLDev( (HDRIVER)     hDriver,
                   (PHLDEV)      &hLDev,
                   (HDEVICE)     hAssocDevice,
                   (PLDEVSTRICT) &LDevStruct );
```

Calling Parameters

hDriver (HDRIVER)
Driver handle of the device driver creating this adapter.

&hLDev (PHLDEV)
Pointer to variable to receive the returned logical device handle.

hAssocDevice (HDEVICE)
Handle of the physical device or adapter that this logical device is aliasing.

Note: This field can also contain an adapter handle that was coerced to an HDEVICE type.

&LDevStruct
Pointer to the following structure:

```
LDEVSTRUCT struct
           {
             PSZ       LDevDescriptName;
             USHORT    LDevFlags;
             USHORT    LDevClass;
             HDEVICE   LDevHDevice;
             PADJUNCT pAdjunctList;
           };
```

LDevDescriptName (PSZ)
Pointer to an ASCIIZ string containing the logical device key and a brief description of the device. For example:

```
"FIXDSK_#  Fixed Disk Drive"
"COMM_#    Serial Port"
```

Up to the first 16 non-blank characters are used as a key to locate this device. The RMCreateLDev services will substitute the "#" character for the device number supplied in the adjunct list.

Refer to Device Adapter Keys for additional information concerning device or adapter keys.

LDevFlags (USHORT)
Currently, there are no flags defined. This field must be initialized to zero by the caller.

LDevClass (USHORT)
Specifies the type of logical device being created. The logical device returned will be made a child of the Resource Manager node for the class of device specified.

Logical Device Type	Class Node Name
LDEV_CLASS_DASD	DASD
LDEV_CLASS_CDROM	CDROM
LDEV_CLASS_SERIAL	SERIAL
LDEV_CLASS_PARALLEL	PARALLEL
LDEV_CLASS_TAPE	TAPE

Refer to Logical View Management for further information.

LDevHDevice
This field must be initialized to zero by the caller. It returns the handle of a physical device indicated by hAssocDevice when a copy of the logical device node is returned by RMGetNodeInfo.

pAdjunctList (PADJUNCT)
A pointer to a linked list of adjunct data structures. The following adjunct structure should be included in the list:

- **ADJ_DEVICE_NUMBER**

  This adjunct contains the zero-based device number for the logical device being created.

Returns

On systems where the Resource Manager driver is not installed, the library interface code will return RMRC_SUCCESS and a logical device handle of -1L.

RMCreateSysName - Obtain a System Name Handle

This service allows a driver to register a system name with the Resource Manager and associate it with a logical device. A system-name handle (HSYSNAME) is returned by this service to identify the system name.

Information about the system name being registered is passed in a SYSNAMESTRUCT.

Calling Sequence

```
rc = RMCreateSysName( (HDRIVER)        hDriver,
                      (PHSYSNAME)      &hSysName,
                      (HLDEV)          hLDevParent,
                      (PSYSNAMESTRUCT) &SysNameStruct );
```

Calling Parameters

hDriver (HDRIVER)
: Driver handle of the device driver creating this adapter.

&hSysName (PSYSNAME)
: Pointer to variable to receive the returned system-name handle.

hLDevParent (HLDEV)
: Handle of the logical device with which the system name is associated, such as the parent of the system name.

&SysNameStruct
: Pointer to the following structure:

```
SYSNAMESTRUCT struct
                {
                  PSZ      SysDescriptName;
                  PADJUNCT pAdjunctList;
                  USHORT   SysFlags;
                  USHORT   Reserved;
                };
```

SysDescriptName (PSZ)
: Pointer to an ASCIIZ string containing the system name key and a brief description of the system name. For example:

```
"C:    Logical DASD Volume"
"COM1: Communications Port"
```

pAdjunctList
: Pointer to a linked list of adjunct data structures.

The following adjunct structure should be included in the list:

- ADJ_DASD_VOL

  This adjunct is used for DASD-type devices to indicate the capacity and file-system type for the drive letter indicated by the system name.

Returns

On systems where the Resource Manager driver is not installed, the library interface code will return RMRC_SUCCESS and a system name handle of -1L.

RMDeAllocResource - Destroy a Resource Handle

This service destroys a resource handle created by RMAllocResource.

Calling Sequence

```
rc = RMDeallocResource ( (HDRIVER)  hDriver
                         (HRESOURCE) hResource );
```

Calling Parameters

```
hDriver
     Handle of the driver when supplied the resource was created.

hResource
     Handle of the resource to be destroyed.
```

Returns

On systems where the Resource Manager driver is not installed, the library interface code will return RMRC_SUCCESS.

RMDestroyAdapterHandle - Destroy an Adapter Handle

This service releases an adapter handle created by RMCreateAdapter.

Calling Sequence

```
rc = RMDestroyAdapter( (HDRIVER)  hDriver
```

Calling Parameters

hDriver
Handle of the driver supplied when the adapter is created.

hAdapter
Handle of the adapter to be destroyed.

Returns

On systems where the Resource Manager driver is not installed, the library interface code will return RMRC_SUCCESS.

Remarks

Although normal driver execution results in adapter-node creation, there are also environments where adapter node destruction is needed. PCMCIA and docking stations are two environments where the adapter-node destruction service is useful.

Destroying an adapter also destroys any child devices associated with the adapter. Any resource handles allocated to the adapter or child devices are released, as well.

RMDestroyDevice - Destroy a Device Handle

This service releases a device handle created by RMCreateDevice. Any resources assigned to the device are released and the device is destroyed.

Calling Sequence

```
rc = RMDestroyDevice( (HDRIVER) hDriver
                      (HDEVICE) hDevice );
```

Calling Parameters

hDriver

Handle of the driver supplied when device was created.

hDevice

Handle of the the device to be destroyed.

Returns

On systems where the Resource Manager driver is not installed, the library interface code will return RMRC_SUCCESS.

RMDestroyDriver - Destroy a Driver Handle

This service releases a driver handle created by RMCreateDriver. If a driver determines it should fail its initialization (such as unload), the driver will issue this call if it had previously issued an RMCreateDriver request.

Issuing this call will delete all devices, adapters, and resource records created under this driver handle.

Calling Sequence

```
rc = RMDestroyDriver( (HDRIVER) hDriver );
```

Calling Parameters

```
hDriver
     Handle of the driver to be destroyed.
```

Returns

On systems where the Resource Manager driver is not installed, the library interface code will return RMRC_SUCCESS.

Remarks

Drivers that intend to unload must still issue the appropriate DevHelp calls to release IRQs and other kernel resources.

RMDestroyLDev - Destroy a Logical Device Handle

This service destroys a logical device handle created by RMCreateLDev. Destroying a logical device also destroys any system names associated with the logical device.

Calling Sequence

```
rc = RMDestroyLDev( (HDRIVER)  hDriver,
                    (HLDEV)    hLDev    );
```

Calling Parameters

```
hDriver (HDRIVER)
     Driver handle of the device driver that created the logical
     device.

hLDev (HLDEV)
     Handle of the logical device to be destroyed.
```

Returns

On systems where the Resource Manager driver is not installed, the library interface code will return RMRC_SUCCESS.

RMDestroySysName - Destroy a System Name Handle

This service destroys a system name handle that was created by RMCreateSysName.

Calling Sequence

```
rc = RMDestroySysName( (HDRIVER)  hDriver,
                       (HSYSNAME) hSysName    );
```

Calling Parameters

```
hDriver (HDRIVER)
     Driver handle of the device driver that created the logical
     device.

hSysName (HSYSNAME)
     Handle of the system name to be destroyed.
```

Returns

On systems where the Resource Manager driver is not installed, the library interface code will return RMRC_SUCCESS.

RMGetNodeInfo - Return Resource Manager Node Information

This service returns the information content of the Resource Manager handle indicated.

Calling Sequence

```
rc = RMGetNodeInfo( (RMHANDLE)          hRMHandle,
                    (PRM_GETNODE_DATA) &NodeInfo,
                    (USHORT)            NodeInfoSize );
```

Calling Parameters

&hRMHandle (RMHANDLE)

Resource Manager handle whose information is to be returned. The following handle types are allowed:

HandleType	Description
HANDLE_TYPE_DRIVER	Driver Handle
HANDLE_TYPE_ADAPTER	Adapter Handle
HANDLE_TYPE_DEVICE	Device Handle
HANDLE_TYPE_LOGDEV	Logical Device Handle
HANDLE_TYPE_SYSDEV	System Name Handle

&NodeInfo (PRM_GETNODE_DATA)

Pointer to a buffer that will contain the Resource Manager node information.

```
RM_GETNODE_DATA struct
                {
                  ULONG          RMNodeSize;
                  RM_NODE        RMNode;
                };
```

RMNodeSize (ULONG)

Total data length, returned in bytes. If the buffer provided is too small to contain the Resource Manager node information, this field will contain the required buffer length.

RMNode (RM_NODE)

A structure containing information about the requested Resource Manager node.

```
RM_NODE struct
        {
          ULONG              VersionInfo;
          ULONG              NodeType;
          RMHANDLE           DriverHandle;

          union
          {
            PADAPTERSTRUCT  pAdapterNode;
            PDEVICESTRUCT   pDeviceNode;
            PLDEVSTRUCT     pLDevNode;
            PSYSNAMESTRUCT  pSysNameNode;
            PDRIVERSTRUCT   pDriver;
          };
          PRESOURCELIST      pResourceList;
        }
```

VersionInfo (ULONG)
Version of the Resource Management driver.

NodeType (ULONG)
The type of node to which the handle provided refers.

pAdapterNode (PADAPTERSTRUCT)
This field contains a pointer to a structure that describes the Resource Manager node. This structure is a copy of the structure that was provided when the node was created.

The pointer type selected from the union should be based on the NodeType value returned.

NodeType	Structure Pointer	Service
RMTYPE_ADAPTER	pAdapterNode	RMCreateAdapter
RMTYPE_DEVICE	pDeviceNode	RMCreateDevice
RMTYPE_LDEV	pLDevNode	RMCreateLDev
RMTYPE_SYSNAME	pSysNameNode	RMCreateSysName
RMTYPE_DRIVER	pDriver	RMCreateDriver

pResourceList (PRESOURCELIST)

Pointer to a structure containing a count and list of resources assigned to this node.

```
PRESOURCELIST struct
              {
                ULONG          Count;
                RESOURCESTRUCT Resource[1];
              }
```

Count (ULONG)

Count of resource structures returned.

Resource [ ] (RESOURCESTRUCT)

Array of resource structures assigned to this node. Refer to RMAllocResource - Obtain a Resource Handle for a description of the RESOURCESTRUCT datatype.

NodeInfoSize (USHORT)

Size of buffer pointed to by &NodeInfo.

Returns

On systems where the Resource Manager driver is not installed, the library interface code will return RMRC_NOT_INSTALLED.

RMHandleToParent - Return a Parent Handle

This service returns the parent handle of the handle provided.

Calling Sequence

```
rc = RMHandleToParent( (RMHANDLE)    hHandle,
                       (PRMHANDLE)   &hParent );
```

Calling Parameters

hHandle (RMHANDLE)
Handle whose parent is to be determined. Valid handle types for this service include:

HandleType	Description
HANDLE_TYPE_ADAPTER	Adapter Handle
HANDLE_TYPE_DEVICE	Device Handle
HANDLE_TYPE_RESOURCE	Resource Handle
HANDLE_TYPE_LOGDEV	Logical Device Handle
HANDLE_TYPE_SYSDEV	System Name Handle

The parent of a resource handle is considered to be the owner of the resource.

&hParent (PRMHANDLE)   Handle of the parent of the specified handle.

Returns

On systems where the Resource Manager driver is not installed, the library interface code will return RMRC_NOT_INSTALLED.

RMHandleToType - Return the Type of Resource Manager Handle

```
Returns the type of Resource Manager handle supplied.
```

Calling Sequence

```
rc = RMHandleToType( (RMHANDLE)    hHandle,
                     (PUSHORT)     &HandleType )
```

Calling Parameters

hHandle (RMHANDLE)
Handle whose type is to be determined.

&HandleType (USHORT)
Pointer to variable to contain the returned handle type. The following handle types can be returned:

HandleType	Description	Service
HANDLE_TYPE_INVALID	Invalid Handle	None
HANDLE_TYPE_DRIVER	Driver Handle	RMCreateDriver
HANDLE_TYPE_ADAPTER	Adapter Handle	RMCreateAdapter
HANDLE_TYPE_DEVICE	Device Handle	RMCreateDevice
HANDLE_TYPE_RESOURCE	Resource Handle	RMAllocResource
HANDLE_TYPE_LOGDEV	Logical Device Handle	RMCreateLDev
HANDLE_TYPE_SYSDEV	System Name Handle	RMCreateSysName

Returns

On systems where the Resource Manager driver is not installed, the library interface code will return RMRC_NOT_INSTALLED.

RMHDevToHLDev - Return Physical Device Associated with Logical Device

This service returns the Logical Device handle (HLDEV) that is associated with the physical device handle indicated.

Calling Sequence

```
rc = RMHDevToHLDev ( (HDEVICE)    hDevice,
                     (HLDEV)      hStartLDev
                     (PHLDEV)     &hLDev       );
```

Calling Parameters

hDevice

Resource Manager handle to the physical device. The logical device handle that is associated with this physical device handle will be returned.

&hStartLDev (HLDEV)

Handle of the logical device at which to start the search. If all logical nodes are to be searched, then HANDLE_LDEV_ROOT should be specified.

&hLDev (PHLDEV)

Pointer to the variable to receive the logical device handle (HLDEV) associated with the physical device specified.

Returns

On systems where the Resource Manager driver is not installed, the library interface code will return RMRC_NOT_INSTALLED.

RMKeyToHandleList - Search for the Specified Adapter/Device/LDev Key

This service searches for Resource Manager nodes that match the key specified. A list of node handles found is returned in the HandleList structure provided by the caller.

Calling Sequence

```
rc = RMKeyToHandleList( (RMHANDLE)    hStartNode,
                        (PSZ)         SearchKey,
                        (PHANDLELIST) &HandleList );
```

Calling Parameters

hStartNode (RMHANDLE)
The handle of the Resource Manager node at which to start the search. This node and all its descendents will be checked. The handle provided may be an adapter handle (HADAPTER) or logical device handle (HLDEV).

The following "pseudohandles" may also be used as a starting point for a search:

Pseudohandle	Nodes Searched
HANDLE_PHYS_TREE	Physical Device Nodes
HANDLE_SYS_TREE	Logical Device Nodes
HANDLE_DEFAULT_SYSBUS	System Bus Nodes (ISA/EISA/Micro Channel)
HANDLE_X_BUS	Planar Bus Nodes
HANDLE_PCI_BUS	PCI Bus Nodes

SearchKey (PSZ)
Pointer to an ASCIIZ string containing the key to be located. If the key supplied ends in an asterisk (*), then all keys that match the characters up to the asterisk will be returned. Key searches are treated as case-insensitive. For example, "FIXDSK_*" will return all fixed-disk logical device handles.

This service does not currently support full (regular pattern matching), for example, only the asterisk is supported.

Adapter key values are chosen at the discretion of the device-driver supplier and are subject to change.

&HandleList (PHANDLELIST)
Pointer to the following structure:

HANDLELIST struct

```
{
  USHORT        cMaxHandles;
  USHORT        cHandles;
  HADAPTER      Handles[1];
};
```

cMaxHandles (USHORT)

Number of handles that can be accepted in the handles array. This field must be set by the caller.

The default HANDLELIST type provides room for one handle. In this case, cMaxHandles must be set to 1. In the following example, 10 handles are expected:

```
#define MAX_HANDLE_COUNT 10

#define HLISTSIZE(c) ( sizeof(HANDLELIST) +  \
                       (c-1) * sizeof(HADAPTER) )

UCHAR HandleList[HLISTSIZE(MAX_HANDLE_COUNT)];

PHANDLELIST pHndList = (PHANDLELIST) HandleList;

pHndList->cMaxHandles = MAX_HANDLE_COUNT;
```

cHandles (USHORT)

Actual handle count found. This field must be initially set to zero by the caller. The number of handles reported by this field can exceed cMaxHandles. If the field exceeds cMaxHandles, the HandleList structure supplied was too small. The count will be set to the number of handles actually found.

Handles[] (HADAPTER)

Array containing the Resource Manager handles that match the specified key.

Returns

On systems where the Resource Manager driver is not installed, the library interface code will return RMRC_NOT_INSTALLED.

RMModifyResources - Modify Adapter or Device Resource Sets

This service allows for modification of resources owned by an existing adapter or device. Resource handles can be deleted or added to an adapter or device.

Calling Sequence

```
rc = RMModifyResources( (HDRIVER)   hDriver,
                        (HADAPTER)  hAdapter,
                        (USHORT)    ModifyAction,
                        (HRESOURCE) hResource);
```

Calling Parameters

hDriver (HDRIVER)
Driver handle of the device driver that created the adapter or device.

&hAdapter (HADAPTER)
Handle of the adapter or device whose resource set is to be modified.

ModifyAction (USHORT)

RM_MODIFY_ADD
Add the resource handle indicated to the adapter or device.

RM_MODIFY_DELETE
Delete the resource handle indicated.

Deleting a resource implicitly causes an RMDeallocResource to occur; for example, the resource handle is no longer valid.

hResource (HRESOURCE)
Handle of the resource to be added or deleted.

Returns

On systems where the Resource Manager driver is not installed, the library interface code will return RMRC_NOT_INSTALLED.

RMParseSCSIInquiry - Build SCSI Device Description

This service is provided as a convenience for device drivers dealing with SCSI devices. It converts SCSI inquiry data for a device into a device key and a description that can be used in RMCreateDevice.

Calling Sequence

```
rc = RMParseScsiInquiry( (PVOID)   &InquiryData,
                         (PSZ)     DescBuffer,
                         (USHORT)  DescBufferSize  );
```

Calling Parameters

&InquiryData (PVOID)

Pointer to a buffer containing SCSI inquiry data for the device.

DescBuffer (PSZ)

Pointer to a buffer that will receive the device key and device description built by this service. This data may be used as the device description field (DevDescriptName) of the DEVICESTRUCT used to create the device.

Refer to RMCreateDevice - Obtain a Device Handle for further information.

DescBufferSize (USHORT)

Size of the DescBuffer, in bytes.

Returns

On systems where the Resource Manager driver is not installed, the library interface code will return RMRC_NOT_INSTALLED.

RMResToHandleList - Return List of Adapter/Device Handles That Own a Resource

This service returns a list of Adapter/Device handles that own the resource indicated.

Calling Sequence

```
rc = RMResToHandleList( (PRESOURCESTRUCT) &ResStruct,
                        (PHANDLELIST)     &HandleList );
```

Calling Parameters

&ResStruct (PRESOURCESTRUCT)
    Pointer to a RESOURCESTRUCT whose owners are to be located. Refer to RMAllocResource - Obtain a Resource Handle for a description of the RESOURCESTRUCT datatype.

&HandleList (PHANDLELIST)
    Structure containing a list of adapter or device handles that include the specified resource. Refer to RMKeyToHandleList - Search for the Specified Adapter/Device/LDev Key for a description of the HANDLELIST datatype.

Returns

On systems where the Resource Manager driver is not installed, the library interface code will return RMRC_NOT_INSTALLED.

RMUpdateAdjunct - Update Adjunct Data Structure

This service updates an existing adjunct structure.

Calling Sequence

```
rc = RMUpdateAdjunct( (HDRIVER)  hDriver,
                      (HDEVICE)  hDevice,
                      (USHORT)   AdjunctIndex,
                      (PADJUNCT) &AdjunctData   );
```

Calling Parameters

```
hDriver (HDRIVER)
     Driver handle of the device driver that created the adjunct data.

hDevice (HDEVICE)
     Device handle with which the adjunct data is associated.

AdjunctIndex (USHORT)
     Index to the adjunct structure to be replaced.

&AdjunctData
     Pointer to an adjunct structure containing replacement data for
     the adjunct indicated.
```

For Resource Manager version 1.1, the size of the replacement adjunct structure must not exceed the size of the existing adjunct.

Returns

On systems where the Resource Manager driver is not installed, the library interface code will return RMRC_NOT_INSTALLED.

Resource Manager IOCtls

RESOURCE.SYS provides two IOCtls that allow a ring-3 application to obtain a "snapshot" of the Resource Management data structures. Obtaining a snapshot of the Resource Management data structures consists of the following two steps:

1. A data structure representing a depth-first traversal of the Resource Manager node structure is obtained.

2. For each node traversed, the following information is provided:

   A Resource Manager handle to access the node.
   The depth of the node in the tree structure.

3. A copy of each Resource Manager node can be obtained by supplying the node handle returned in Step 1.

The two IOCtls for Resource Manager are:

- Get Resource Manager Node Data - Function 01h

- Enumerate Resource Manager Nodes - Function 02h

Get Resource Manager Node Data - Function 01h

Category 80h, Function 01h

Category:
  80h
Function:
  01h
Description:
  Get Resource Manager Node Data

Select an item:

Description
Parameter Packet Format
Data Packet Format
Remarks

Description - Category 80h, Function 01h

This function returns the contents of the Resource Manager node indicated by the handle provided.

RMHandle - Category 80h, Function 01h

RMHandle

This field must be initialized to the handle of the Resource Manager node to be interrogated.

Parameter Packet Format - Category 80h, Function 01h

Field	Length
RMHandle	ULONG

RMNodeSize - Category 80h, Function 01h

RMNodeSize
Size of the Resource Manager node information returned.

RMNode - Category 80h, Function 01h

RMNode

This field is set to a structure describing the Resource Manager node and its associated resources.

```
RM_NODE struct

    {

      ULONG            VersionInfo;
      ULONG            NodeType;
      RMHANDLE         DriverHandle;

      union
      {
        PADAPTERSTRUCT  pAdapterNode;
        PDEVICESTRUCT   pDeviceNode;
        PLDEVSTRUCT     pLDevNode;
        PSYSNAMESTRUCT  pSysNameNode;
        PDRIVERSTRUCT   pDriver;
      };
      PRESOURCELIST    pResourceList;
    }
```

VersionInfo (ULONG)

Version of the Resource Management driver.

NodeType (ULONG)

The type of node to which the handle provided refers.

pAdapterNode (PADAPTERSTRUCT)

This field contains a pointer to a structure that describes the Resource Manager node. This structure is a copy of the structure that was provided when the node was created. The pointer type selected from the union should be based on the NodeType value returned.

NodeType	Structure Pointer	Service
RMTYPE_ADAPTER	pAdapterNode	RMCreateAdapter

RMTYPE_DEVICE	pDeviceNode	RMCreateDevice
RMTYPE_LDEV	pLDevNode	RMCreateLDev
RMTYPE_SYSNAME	pSysNameNode	RMCreateSysName
RMTYPE_DRIVER	pDriver	RMCreateDriver

pResourceList (PRESOURCELIST)

Pointer to a structure containing a count and list of resource assigned to this node.

PRESOURCELIST struct

```
{
  ULONG          Count;
  RESOURCESTRUCT Resource[1];
}
```

Count (ULONG)

Count of resource structures returned.

Resource[] (RESOURCESTRUCT)

An array of resource structures assigned to this node. Refer to RMAllocResource - Obtain a Resource Handle for a description of the RESOURCESTRUCT datatype.

Data Packet Format - Category 80h, Function 01h

All data packet fields are output fields for this function.

Field	Length
RMNodeSize	ULONG
RMNode	

Remarks - Category 80h, Function 01h

None.

Enumerate Resource Manager Nodes - Function 02h

Category 80h, Function 02h

Category:
80h
Function:
02h
Description:
Enumerate Resource Manager Nodes

Select an item:

Description
Parameter Packet Format
Data Packet Format
Remarks

Description - Category 80h, Function 02h

This function traverses the Resource Manager node structure and returns the results of the traversal as a list of Resource Manager handles and traversal depth.

Command - Category 80h, Function 02h

Command

Traverses the physical device tree. This traversal reports all adapters and devices registered with the Resource Manager.

Parameter Packet Format - Category 80h, Function 02h

This field must indicate the type of traversal being requested.

Field	Length
Command	WORD

NumEntries - Category 80h, Function 02h

NumEntries

This field reports the number of node entries traversed.

NodeEntry[] - Category 80h, Function 02h

NodeEntry[]

This field is an array of the following structure:

```
NODEENTRY struct
          {
           RMHANDLE RMHandle;
           ULONG    Depth;
          };
```

RMHandle (RMHANDLE)

Resource Manager handle of the node traversed.

Depth (ULONG)

Level of the tree structure on which the node resides.

Data Packet Format - Category 80h, Function 02h

All data packet fields are output fields for this function.

Field	Length
NumEntries	ULONG
NodeEntry[]	8 * NumEntries

Remarks - Category 80h, Function 02h

None.

RMBASE.H

```
/*************************************************************************
/*
/* COPYRIGHT    Copyright (C) 1992 IBM Corporation
/*
/*    The following IBM OS/2 2.1 source code is provided to you solely for
/*    the purpose of assisting you in your development of OS/2 2.x device
/*    drivers. You may use this code in accordance with the IBM License
/*    Agreement provided in the IBM Device Driver Source Kit for OS/2. This
/*    Copyright statement may not be removed.
/*
/*************************************************************************
/*************************************************************************
 *
 * SOURCE FILE NAME =  RMBASE.H
 *
 * DESCRIPTIVE NAME =  RM Base types and definitions
 *
 *
 *
 * VERSION = V1.01
 *
 * DATE
 *
 * DESCRIPTION
 *
 * Purpose
 *
 *
 *
 * FUNCTIONS
 *
 * NOTES
 *
 *
 * STRUCTURES
 *
 * EXTERNAL REFERENCES
 *
 *
 *
 * EXTERNAL FUNCTIONS
 *
*/


#ifndef __RM_HEADER__
#define __RM_HEADER__


#define CMVERSION_MAJOR    0x01
#define CMVERSION_MINOR    0x01

typedef ULONG    RMHANDLE, FAR *PRMHANDLE, NEAR *NPRMHANDLE;
typedef RMHANDLE HDRIVER;
typedef RMHANDLE HADAPTER;
typedef RMHANDLE HDEVICE;
typedef RMHANDLE HRESOURCE;
typedef RMHANDLE HLDEV;
typedef RMHANDLE HSYSNAME;

typedef HDRIVER   FAR *PHDRIVER;
typedef HDRIVER   NEAR *NPHDRIVER;
typedef HADAPTER  FAR *PHADAPTER;
typedef HADAPTER  NEAR *NPHADAPTER;
typedef HDEVICE   FAR *PHDEVICE;
typedef HDEVICE   NEAR *NPHDEVICE;
typedef HRESOURCE FAR *PHRESOURCE;
typedef HRESOURCE NEAR *NPHRESOURCE;
typedef HLDEV     FAR *PHLDEV;
typedef HLDEV     NEAR *NPHLDEV;
typedef HSYSNAME  FAR *PHSYSNAME;
typedef HSYSNAME  NEAR *NPHSYSNAME;


/*************************************************************************
/*
/* Driver Structure
/*
/*************************************************************************

typedef struct {
   USHORT Year;
   UCHAR  Month;
   UCHAR  Day;
} DATESTAMP, FAR *PDATESTAMP, NEAR *NPDATESTAMP;


/* Callback for Grant/Yield; Indicates CM resolves resource to handle */
#ifdef __IBMC__
typedef ULONG PFNRMCB;
#else
typedef USHORT (_cdecl FAR *PFNRMCB)(HRESOURCE hResource);
#endif


typedef struct {
   PSZ       DrvrName;
   PSZ       DrvrDescript;
   PSZ       VendorName;
   UCHAR     MajorVer;
   UCHAR     MinorVer;
   DATESTAMP Date;
   USHORT    DrvrFlags;
   USHORT    DrvrType;
   USHORT    DrvrSubType;
   PFNRMCB   DrvrCallback;     /* Event notification */
} DRIVERSTRUCT, FAR *PDRIVERSTRUCT, NEAR *NPDRIVERSTRUCT;



/********************************/
/* pDriverStruct->DriverFlags */
/********************************/

#define DRF_DYNAMIC  0x0001
#define DRF_STATIC   0x0000

/* pDriverStruct->DriverType      */
/*   pDriverStruct->DriverSubType */
#define DRT_UNDEFINED        0
  #define DRS_UNDEFINED      0

#define DRT_PCMCIA           1
  #define DRS_SOCKETSERV     1
  #define DRS_CARDSERV       2
  #define DRS_CLIENT         3
```

```
#define DRT_ADDDM            2
  #define DRS_DM             1
  #define DRS_FILTER         2
  #define DRS_ADD            3
  #define DRS_DM_TRANSPORT   4

#define DRT_OS2              3
  #define DRS_CHAR           1
  #define DRS_BLOCK          2
  #define DRS_APP_HELPER     3

#define DRT_NETWORK          4
#define DRT_VIDEO            5
#define DRT_AUDIO            6
#define DRT_SERVICE          7



/*************************************************************************
/*
/* Adapter Structure  - Device Bus
/*
/*************************************************************************

typedef struct {
   USHORT ADDHandle;
   USHORT UnitHandle;
}ADD_UNIT, FAR *PADD_UNIT, NEAR *NPADD_UNIT;

typedef struct {
   USHORT VolFlags;
   USHORT VolIFSType;
   ULONG  VolSize;
   ULONG  VolID;
} DASD_VOL, FAR *PDASD_VOL, NEAR *NPDASD_VOL;

typedef struct _ADJUNCT FAR *PADJUNCT;
typedef struct _ADJUNCT{
   struct _ADJHEADER {
     PADJUNCT pNextAdj;
     USHORT   AdjLength;
     USHORT   AdjType;
   };
   union {
     USHORT           AdjBase;
     USHORT           SCSI_Target_LUN;
     USHORT           Adapter_Number;
     USHORT           Device_Number;
     USHORT           PCI_DevFunc;
     USHORT           Model_Info;
     ADD_UNIT         Add_Unit;
     DASD_VOL         Dasd_Vol;
   };
}ADJUNCT, NEAR *NPADJUNCT;



/********************************/
/* pAdjunct->AdjunctType        */
/********************************/

#define ADJ_HEADER_SIZE        sizeof(struct _ADJHEADER)

#define ADJ_SCSI_TARGET_LUN    1
#define ADJ_ADAPTER_NUMBER     2
#define ADJ_DEVICE_NUMBER      3
#define ADJ_PCI_DEVFUNC        4
#define ADJ_MODEL_INFO         5
#define ADJ_ADD_UNIT           6
#define ADJ_DASD_VOL           7


typedef struct{
   PSZ          AdaptDescriptName;
   USHORT       AdaptFlags;
   USHORT       BaseType;             /* From PCI/PNP */
   USHORT       SubType;
   USHORT       InterfaceType;
   USHORT       HostBusType;
   USHORT       HostBusWidth;
   PADJUNCT     pAdjunctList;
   ULONG        Reserved;             /* Logical Name addition? */
} ADAPTERSTRUCT, FAR *PADAPTERSTRUCT, NEAR *NPADAPTERSTRUCT;
```

```
/********************************/
/* pAdapteStruct->BaseType      */
/* pAdapteStruct->Sub           */
/* pAdapteStruct->InterfaceType */
/* From PNP/PCI Specs           */
/********************************/

#define AS_BASE_RESERVED          0x00
 #define AS_SUB_OTHER             0x80      /* Can Be used by any BASE type
  #define AS_INTF_GENERIC         0x01      /* Can Be used by any SUB type

#define AS_BASE_MSD               0x01      /* Mass Storage Device
 #define AS_SUB_SCSI              0x01
 #define AS_SUB_IDE               0x02
 #define AS_SUB_FLPY              0x03
 #define AS_SUB_IPI               0x04

#define AS_BASE_NETWORK           0x02     /* Network Interface Controller
 #define AS_SUB_ETHERNET          0x01
 #define AS_SUB_TOKENRING         0x02
 #define AS_SUB_FDDI              0x03

#define AS_BASE_DISPLAY           0x03     /* Display Controller
 #define AS_SUB_VGA               0x01
  #define AS_INTF_VGA_GEN         0x01
  #define AS_INTF_VESA_SVGA       0x02
 #define AS_SUB_XGA               0x02


#define AS_BASE_MMEDIA            0x04     /* Multi-media Controller
 #define AS_SUB_MM_VIDEO          0x01
 #define AS_SUB_MM_AUDIO          0x02

#define AS_BASE_MEMORY            0x05    /* Memory
```